

ME-330

ME 330: Mechatronics – Laboratory 5

Closed Loop Servo Controller

Nicholas Agtual, Dalaney Conte, Evaline Cuatok, Josh Gipson

Department of Mechanical Engineering

Spring 2021

**Abstract:**

The Closed Loop Servo Controller Lab develops on the control system introduced in the previous lab. A proportional, derivative, and integral position controller are used to minimize the error between the input setpoint position and recorded output position of a motor. The proportional and integral controls were tested by individually altering their constants, and then tested together by changing their variables to model a PI controller. The last section of the lab utilizes all three types of controls by including the derivative position controller, and the system was tuned to achieve desired harmonic motion for the output. The final portion gave insight on how a PID controller functions and the challenges to compute the correct combination of constant values for a specified damping condition.

Contents

[1 Introduction and Objectives 5](#_Toc101987479)

[1.1 Objectives 5](#_Toc101987480)

[1.2 Required Components 5](#_Toc101987481)

[2 Proportional Position Controller 7](#_Toc101987482)

[2.1 Physical Circuit 7](#_Toc101987483)

[2.2 Arduino Code 8](#_Toc101987484)

[2.3 Lab Questions 10](#_Toc101987485)

[3 Derivative Position Controller 14](#_Toc101987486)

[3.1 Physical Circuit 14](#_Toc101987487)

[3.2 Arduino Code 14](#_Toc101987488)

[3.3 Lab Questions 14](#_Toc101987489)

[4 System Modeling 17](#_Toc101987490)

[4.1 Lab Questions 20](#_Toc101987491)

[5 Proportional Integral Position Controller 20](#_Toc101987492)

[5.1 Physical Circuit 20](#_Toc101987493)

[5.2 Arduino Code 20](#_Toc101987494)

[5.3 Lab Questions 20](#_Toc101987495)

[6 Proportional Integral Derivative Position Controller 22](#_Toc101987496)

[6.1 Physical Circuit 22](#_Toc101987497)

[6.2 Arduino Code 22](#_Toc101987498)

[6.3 Lab Questions 23](#_Toc101987499)

[8 Appendices 26](#_Toc101987500)

[9 References 27](#_Toc101987501)

List of Figures

[Figure 1: Arduino Uno Microcontroller 4](#_Toc101815515)

[Figure 2: Potentiometer 5](#_Toc101815516)

[Figure 3: L298N Motor Drive Controller 6](#_Toc101815517)

[Figure 4: Step Input/Output (Derivative Constant - 0.5) 9](#_Toc101815518)

[Figure 5: Step Input/Ouput (Derivative Constant -2) 9](#_Toc101815519)

[Figure 6: Step Input/Output (Derivative Constant - 5) 10](#_Toc101815520)

[Figure 7: Step Input/Output (Derivative Constant - 15) 10](#_Toc101815521)

[Figure 8: Estimated Transfer Function and Fit Estimation 12](#_Toc101815522)

[Figure 9: MATLAB Control System Designer 13](#_Toc101815523)

[Figure 10: Suggested PI Constants 13](#_Toc101815524)

[Figure 11: PI Controller Response 15](#_Toc101815525)

List of Tables

[Table 1: PI Controller Constants 14](#_Toc101815526)

# 1 Introduction and Objectives

## 1.1 Objectives

The objective of this lab is to control the output position of a servo motor by using three different controllers: proportional, derivative, and integral position controllers. A circuit incorporating a servo motor and potentiometer is constructed and modified by programming on the Arduino. Once a closed-loop PID controller is implemented, it is tuned to achieve different system responses. After those systems are defined, improvements to the circuit and programming are considered.

## 1.2 Required Components

Most of the items used for the constructed circuit have been introduced in previous labs. The main elements used are as follows:

**Arduino UNO Microcontroller**

**A close-up of a circuit board

Description automatically generated with medium confidence**

Figure 1: Arduino Uno Microcontroller. Adapted from ref. (7).

The Arduino UNO microcontroller is an electronics platform that reads inputs and transforms it into an output. This was altered in the lab by editing and uploading instructions utilizing the accompanying Arduino program to control isolated portions of the circuit.

**Potentiometers (Setpoint and Servo)**

Diagram

Description automatically generated

Figure 2: Potentiometers. Adapted from ref. (7).

Two potentiometers were utilized in the circuit. The servo potentiometer connected to the DC motor is used to record the angular position of the motor to display the output of the system. The other potentiometer is used to focus on the input by setting a desired setpoint position.

**L298N Motor Drive Controller**

Graphical user interface, schematic

Description automatically generated

Figure 3: L298N Motor Drive Controller. Adapted from ref. (7).

The motor drive controller is used to control the direction of the motor. The logic pins of the motor control the forward/reverse directions by changing the polarity of the input voltage. This component is used in place of an H-bridge, which would perform the same outcome by wiring a combination of transistors and diodes.

# 2 Proportional Position Controller

A proportional position controller evaluates the input of the setpoint position and produces an output that is proportional to the error between the two values. By determining the gain factor “” of the system, the value of the error can be determined. When using a proportional controller there will always be a guaranteed error between the input and output in order to generate the proportional response.

## 2.1 Physical Circuit

The process to build the physical circuit for this lab involved looking at the schematically assembled circuit provided in the lab instructions and following the paths of the wires to each component. As seen in Figure 4, each wire in the circuit was labeled and relatively simple to follow. The circuit was built by starting on the left side of the schematic and working towards the connection of the power and ground of the breadboard. Troubleshooting only occurred when wire was overlooked and not connected completely, causing the motor to run improperly. After the connection was wired securely the circuit performed as intended and was used for all five sections of the lab.

Diagram

Description automatically generated

Figure 4: Schematic of Completed Circuit. Adapted from ref. (7).

The input circuits and the proper pin connections of the potentiometers were given and can be seen in Figure 5. Both grounds were in reference to the breadboard and their source voltages were powered by the Arduino. Both were configured in a voltage divider circuit.

Diagram

Description automatically generated

Figure 5: Input Circuits of Potentiometers. Adapted from ref. (7).

## 2.2 Arduino Code

#define POT1 A0 //servo potentiometer pin

#define POT2 A1 //input potentiometer pin

#define DIRA 7  //Direction pin

#define DIRB 8  //Direction pin

#define ENABLE 9 //Enable pin

float error=0;            //Define error variable

float previous\_error=0;   //Define previous error for derivative

float integral=0;         //Define integral variable

float derivative=0;       //Define derivative variable

float proportional=0;

float dt=0;

float tic=0;

float toc=0;

float Kd=0;   //Derivative Constant

float Kp=3.57;   //Proportional Constant

float Ki=0.0;   //Integral Constant

float PWM=0;

void setup() {

 pinMode(DIRA, OUTPUT);

 pinMode(DIRB, OUTPUT);

 pinMode(ENABLE, OUTPUT);

 pinMode(POT1, INPUT);

 pinMode(POT2, INPUT);

**Serial**.begin(9600);

 TCCR1B = TCCR1B & B11111000 | B00000001; // Changes the PWM to 31kHz

}

void loop() {

previous\_error=error;

 int position\_servo = analogRead(POT1);

 int position\_input = analogRead(POT2);

 error=position\_input-position\_servo;      //compute position error

 integral=integral+Ki\*error\*dt;            // compute integral

 derivative=(error-previous\_error)\*Kd/dt;  // compute derivative

 proportional=Kp\*error;                    // compute proportional

 PWM=proportional+derivative+integral;     // input to plant computed from PID controller

 dt=(tic-toc)/1000.0;                       //time in seconds

 toc=tic;

 tic=millis();

 if (PWM>255)                              // limit PWM output

   PWM=255;

 if (PWM<-255)

   PWM=-255;

 if (PWM>0){                               // Set motor direction to motor controller

   digitalWrite(DIRA,1);

   digitalWrite(DIRB,0);

   analogWrite(ENABLE,PWM);

 }

 else{

   digitalWrite(DIRA,0);

   digitalWrite(DIRB,1);

   analogWrite(ENABLE,-PWM);

 }

// Print relevant data

**Serial**.print(error);

**Serial**.print(" ");

**Serial**.print(previous\_error);

**Serial**.print(" ");

**Serial**.print(integral);

**Serial**.print(" ");

**Serial**.print(derivative);

**Serial**.print(" ");

**Serial**.print(PWM);

**Serial**.print(" ");

**Serial**.print(dt);

**Serial**.print(" ");

**Serial**.print(position\_servo);

**Serial**.print(" ");

**Serial**.println(position\_input);

}

## 2.3 Lab Questions

1. Increase and decrease the proportional constant (e.g. 1.5, 2.5, 5.5, 15.0, 20). Follow the same steps 2-12 of the section 2.4 and discuss what you observe.

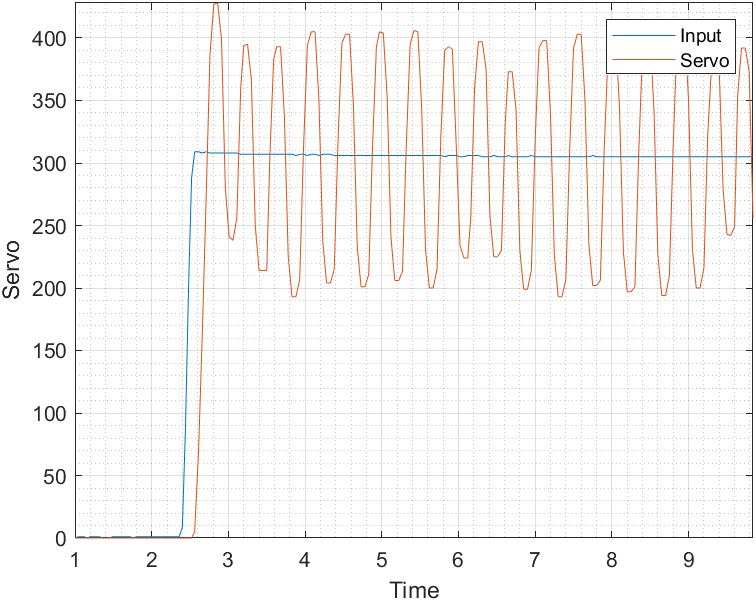


Figure 6: Step Input/Output (Proportional Constant - 10)

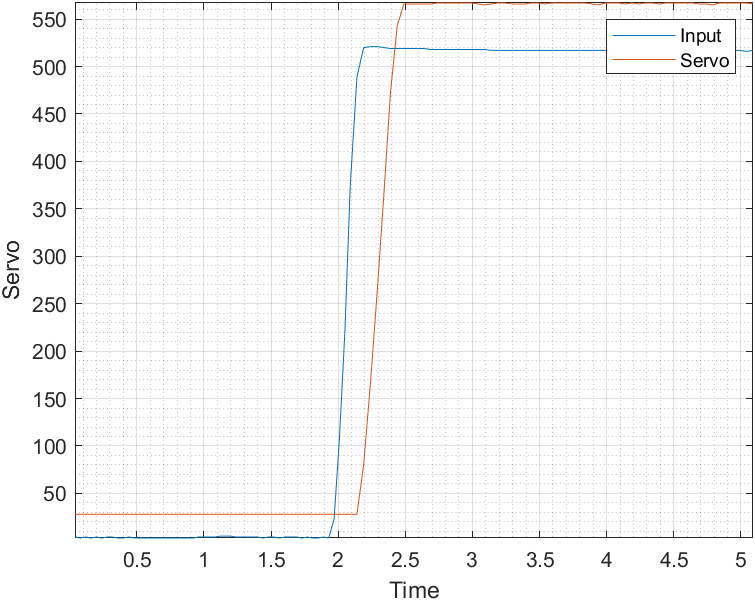


Figure 7: Step Input/Output (Proportional Constant - 1.5)

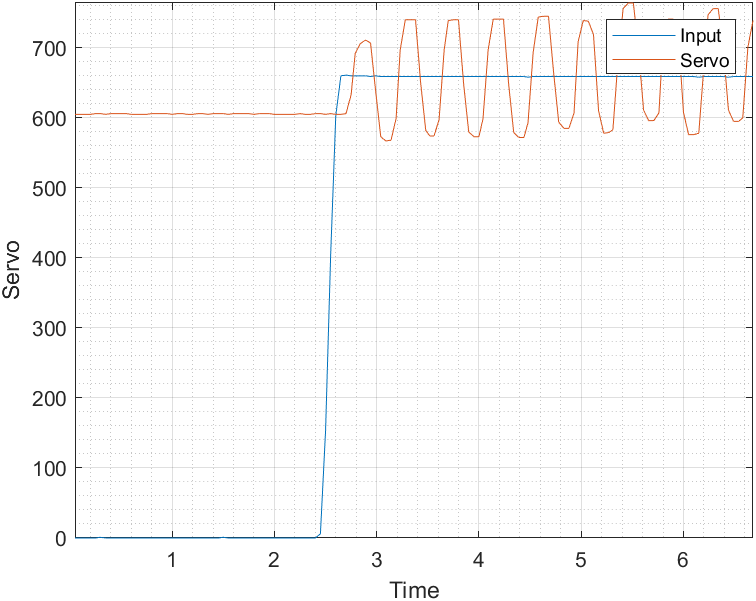


Figure 8: Step Input/Output (Proportional Constant - 5)

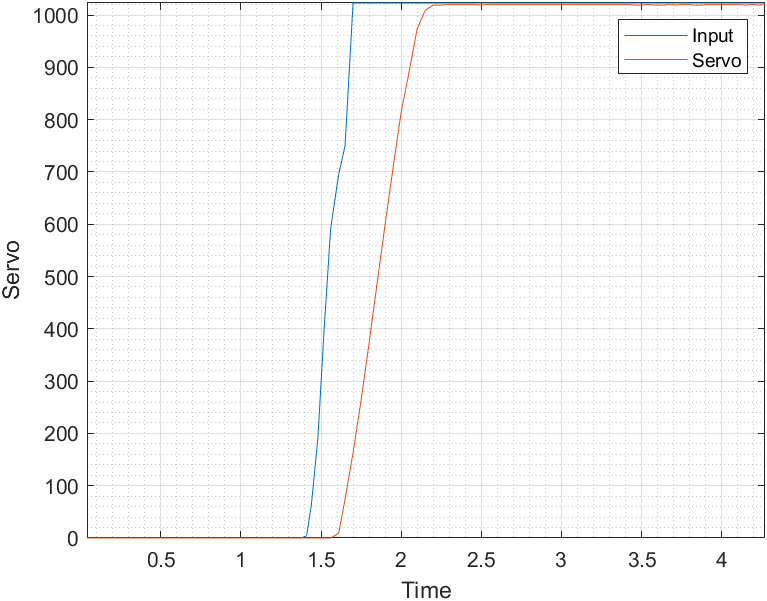


Figure 9: Step Input/Output (Proportional Constant - 3.55)

The results displayed above show the behavior of the servo motor as the proportional constant was adjusted to various values. The input and output tended to show similar trends. As the proportional constant was increased, the oscillations around the steady state value of the input increased as well.

1. Find the proportional constant threshold at which the output is oscillatory. Save the data for this case.

From the results, the threshold in which the output was oscillatory was 3.55.

Graphical user interface, application, table

Description automatically generated

Figure 10: Step Input/Output (Proportional Constant - 3.57)

As seen in the graph, the output began to oscillate at a value greater than 3.55 but began to converge to equilibrium behavior signifying it was close to the threshold.

1. Based on the results of question 1, what system type is this?

The results derived from question 1 display behavior of a Type 0 system. As the system began to oscillate it displayed the behavior of a static error constant that remains constant but with opposite peaks. However, this does not display a perfect example of this system as there is some discrepancy in the peak values, but for the most part they are oscillating at a similar amount around the setpoint position.

1. Review Appendix A, theoretically what system type is this?

Theoretically the system is Type 0 due to it being a step input and the proportional constant is kept at a constant value while the system performs and attempts to reach equilibrium. A Type 0 system with a step input will also have an error that is constant which should occur with the proportional position controller.

1. If the actual and theoretical systems are not the same explain why the discrepancy exists.

Discrepancies between the systems can be attributed to noise within the circuit and the equipment being used not being able to perform to ideal standards. If the signals were disrupted while traveling to and from the inputs and outputs, then the trends that were displayed would reveal those disruptions. In addition, oxidation that occurs on the windings of the potentiometer can lead to residual noise.

# 3 Derivative Position Controller

## 3.1 Physical Circuit

Reference the description of the physical circuit in section 2.1

## 3.2 Arduino Code

The Arduino code for the derivative controller is the same as the code seen in 2.2 Arduino Code, with the following changes:

float Kd=15;   //Derivative Constant

float Kp=0;   //Proportional Constant

float Ki=0.0;   //Integral Constant

Note that the non-zero constants were subject to change.

## 3.3 Lab Questions

1. Increase and decrease the derivative constant. Follow the same steps 2-12 of section 2.4 and discuss what you observe.

Graphical user interface, chart, application, table, Excel

Description automatically generated

Figure 11: Step Input/Output (Derivative Constant - 0.5)

Graphical user interface, chart, application, table, Excel

Description automatically generated

Figure 12: Step Input/Output (Derivative Constant -2)

Graphical user interface, chart, application, table, Excel

Description automatically generated

Figure 13: Step Input/Output (Derivative Constant - 5)

Graphical user interface, chart, application, table, Excel

Description automatically generated

Figure 14: Step Input/Output (Derivative Constant - 15)

After doing multiple trials with different Derivative constants, our results can be seen above. The output of the servo seems to be a lot more consistent than our input. The Servo’s output ADC reading ranges from 150-200 as it plateaus. With our input the plateau seems to have arrange from 300-600 ADC. As we changed the derivative constant to 15 our input and output seemed to be closest. As the derivative constant rose it seems our output seems to be less stable but with a lower derivative constant the output seems to be more stable.

1. Based on the results of question 1, what system type is this?

Based on our results I would say that this is a type 1 system as the input and output change together at almost the same velocity. There is some variation, but I believe this noise in the system could account for this.

1. Review Appendix A, theoretically what system type is this?

The derivative position controller monitors the rate of change of the process and changes the output to account for random activity. The system should be a type 0 as it can involve step and ramp but in our case we only see the ramp

1. If the actual and theoretical systems are not the same explain why the discrepancy exists.

The discrepancy can be due to the noise in the circuit. The difference could also be due to the input not having a lot of variation and so the output only had ramp involved. The input could only involve ramp, so the output doesn’t hold step as well.

# 4 System Modeling

Utilizing MATLAB, the response data for the derivative controller was used to suggest gains for a proportional integral controller. With the data from the derivative controller, the MATLAB “tfest” function was used to estimate the corresponding transfer function. The system was specified to have 2 poles and no zeroes. MATLAB determines a fit estimation, and it was required that this estimation be higher that 85%. See Figure 15 for the transfer obtained from the derivative output data as well as the fir estimation, which ended up being 94.13%.

Text

Description automatically generated

Figure 15: Estimated Transfer Function and Fit Estimation

Using the predefined MATLAB function, “tf”, and adding back in the forward integral, a transfer function was generated. For this function, the coefficients of the estimated transfer function without the forward integral were used. Using MATLAB’s control system designer, the PID constants were able to be estimated given the transfer function that was calculated. The control system designer can be seen in Figure 16.

Graphical user interface, application

Description automatically generated

Figure 16: MATLAB Control System Designer

Finally, the PI constants can be seen in Figure 17.

Graphical user interface, text, application

Description automatically generated

Figure 17: Suggested PI Constants

## 4.1 Lab Questions

1. Report on the results you obtained.

After completing this portion of the report, it was clear that the input data affected the recommended tuning constants. On the first attempt, data from a derivative controller test was used and it resulted in a fit estimation below 85%. The data used in this case was a derivative controller that did not perform desirably, i.e., a lot of overshoot and oscillations. Once the data from a better test was used, the fit estimation increased.

# 5 Proportional Integral Position Controller

## 5.1 Physical Circuit

Refer to Section 2.2 for the physical circuit configuration.

## 5.2 Arduino Code

The Arduino code for the derivative controller is the same as the code seen in 2.2 Arduino Code, with the following changes:

float Kd=0;   //Derivative Constant

float Kp=1;   //Proportional Constant

float Ki=3;   //Integral Constant

Note that the non-zero constants were subject to change.

## 5.3 Lab Questions

1. Discuss and interpret your observations from changing the PI gains.

The constants found in 4 System Modeling were used as a baseline for the PI controller. Further tuning was necessary to optimize the response. Figure 18 display different combinations of PI constants, with the values being shown in Table 1.

Table 1: PI Controller Constants

|  |  |  |
| --- | --- | --- |
|  |  |  |
| **Top Left** | 1 | 3 |
| **Top Right** | 4.1876 | 4.1876 |
| **Bottom Left** | 3 | 3 |
| **Bottom Right** | 1.33 | 3 |

Chart, line chart, box and whisker chart

Description automatically generated

Figure 18: PI Controller Response

For this PI controller, it was evident that with the lack of the derivative portion, minimizing overshoot is difficult. It also became clear that a lower proportional constant resulted is a response with fewer oscillations. The integral constant performed best when it was higher than the proportional constant.

1. Based on the results of question 1, what system type is this?

Based on the results of question 1, this system is type 1 because the output starts at zero and reaches a constant value.

1. Theoretically, what system type is this?

Theoretically, this is a type 1 system because of the roots, none lie on the origin. See Figure 19 for a graphical representation on a root locus plot.

Chart

Description automatically generated

Figure 19: Root Locus for PI Controller

1. If the actual and theoretical systems are not the same, explain why the discrepancies exist.

The theoretical system and actual system type are the same.

# 6 Proportional Integral Derivative Position Controller

The Proportional Integral Derivative Position Controller, or PID Controller, is a type of controller that controls position with a feedback mechanism that helps to control or limit the amount of error. Because there will be different amounts of error with each process, the PID is able to determine how much correction is needed. This is done by using different amounts of proportional, derivative, and integral action. All three constants will be added together to produce the output signal.

## 6.1 Physical Circuit

The physical circuit used in this section is the same as the circuit built in section 2.2.

## 6.2 Arduino Code

The Arduino code for this section of the lab is the same as the Arduino code from section 2.2 with the values being nonzero.

float Kd=1;   //Derivative Constant

float Kp=10;   //Proportional Constant

float Ki=1;   //Integral Constant

## 6.3 Lab Questions

1. Discuss the challenges and results from your tuning experimentation.

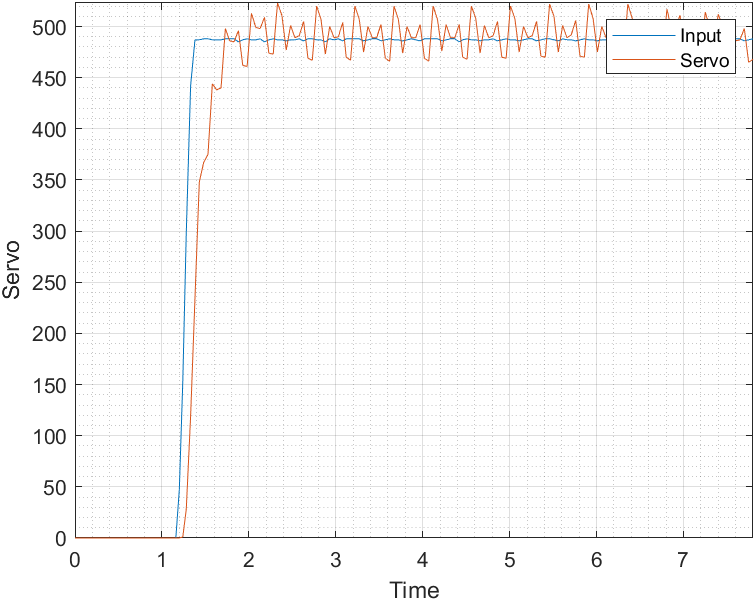


Figure 20: Critically Damped Response (K = 1, Kd = 10, Ki = 1)

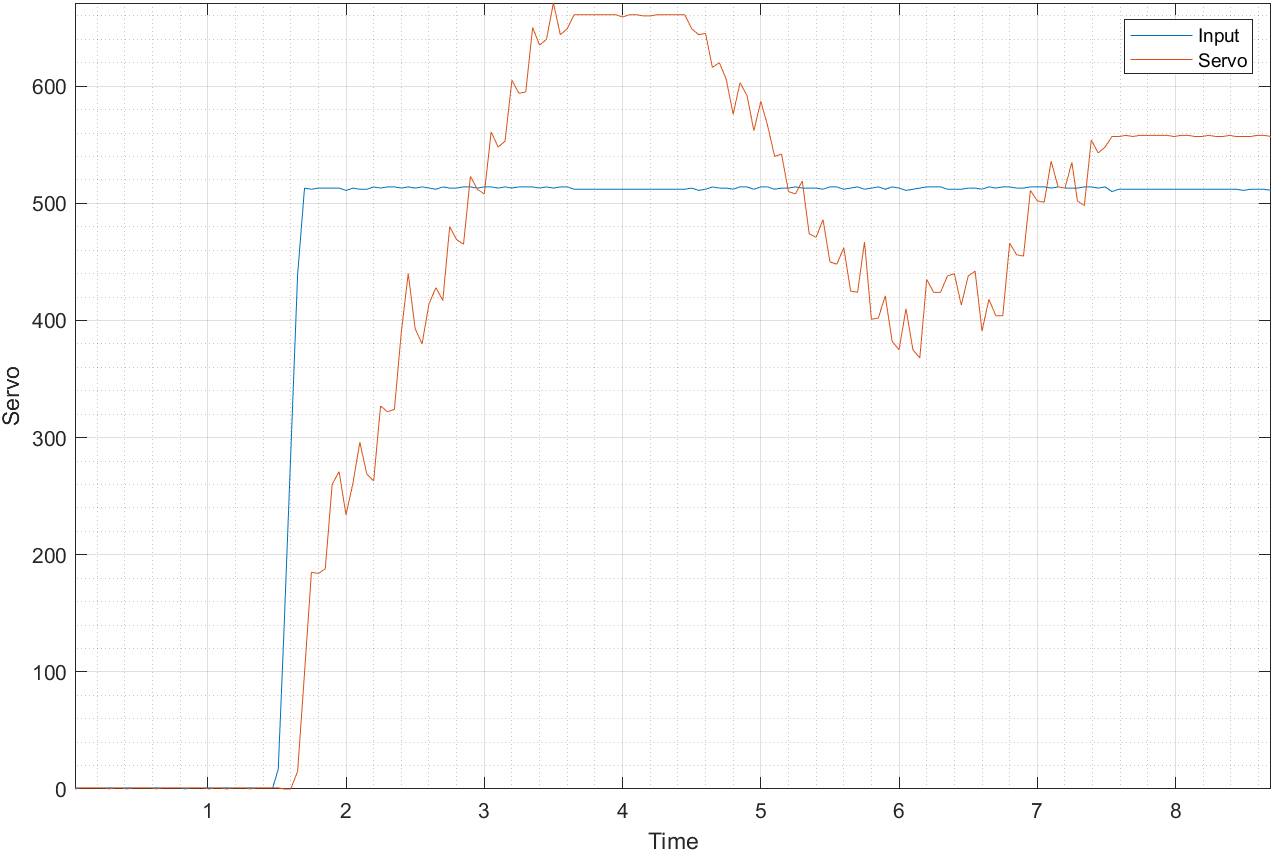


Figure 21: Underdamped Response (Kp = 3, Kd = 3; Ki = 1)

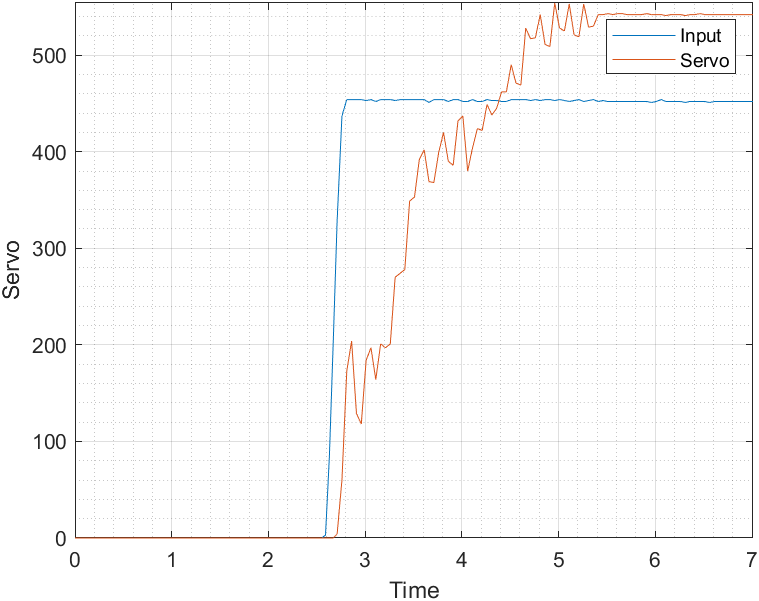


Figure 22: Overdamped Response (Kp = 1, Kd = 1, Ki = 1)

Throughout the tuning experimentation, it was very difficult to get our graphs to look like those modeled as critically damped, underdamped, and overdamped. As can be seen in Figure 20, the graph is critically damped, however there are a lot of oscillations. In order to make the graph smoother without oscillations, we tried to change the values of each constant slightly but that ended up producing an entirely different graph. When trying to produce an underdamped response, we ended up with the graph seen in Figure 21. The response was slow to get to the setpoint and after it overshot the setpoint, it went back down below the setpoint before overshooting again. We adjusted the values again to prevent the response from undershooting, but the delay time became slower. Lastly, for the overdamped response, our graph is shown in Figure 22. The response was slow to catch up to the setpoint however it never converged at the setpoint. Instead, it went past the setpoint and overshot. Overall, it was difficult for us to find a balance when changing the values for each constant because changing the values too much from what they were originally would cause the graph to either overshoot too much or undershoot by too much.

1. What tuning methodology do you recommend for the system?

The tuning methodology that would be recommended for the system is to first try out a few combinations of values for the proportional, derivative, and integral constant and then see what the resulting graphs look like. Depending on what is trying to be achieved (critically damped, overdamped, or underdamped), these values can be adjusted accordingly. According to Appendix C, the proportional constant should be adjusted first because it can be used to offset the error. However, there must already be an error to generate a proportional response. Next the integral constant should be changed because it will eliminate any error after the application of the proportional constant. Once there is no more error, the integral term will no longer grow. Lastly the value for the derivative constant should be changed if there are too many oscillations because it flattens the error trajectory. However, this only works if the proportional and integral constants are already minimizing the error.

7 Conclusion

From this experiment, we were able to learn more about programming Arduino in order to perform circuit analyses. Additionally, this experiment allowed for a better understanding of the proportional integral derivative controller, specifically how to tune it for a particular response time and how to compute the transfer function. When building the circuit for section 2, the proportional position controller, we had some difficulty getting the correct results since the graph was not correct. In order to fix this, we had to recheck our circuit and make sure that all of the parts were connected correctly. Once that was done, we had to adjust the code a few times to make sure that there were no errors. Other than getting the initial setup of both the physical circuit and the Arduino code working, we also had difficulty with section 6, the PID controller as we could not figure out how to adjust the values of the constants to get us the proper graph for a critically damped, underdamped, and overdamped response. Despite these difficulties, the analysis of the circuit and control system were completed successfully.

# 8 Appendices

**MATLAB CODE:**

Plotter:

close all;

SECTION 1

results = zeros(2,8);

SECTION 2

% Separate the result columns

current\_error = results(:,1);

previous\_error = results(:,2);

integral = results(:,3);

derivative = results(:,4);

PWM = results(:,5);

dt = results(:,6);

pos\_servo = results(:,7);

pos\_input = results(:,8);

% Create the time vector

time = zeros(size(dt));

time(1) = dt(1);

for i = 2:length(dt)

time(i) = time(i-1) + dt(i);

end

plot(time,pos\_input,time,pos\_servo); axis tight; grid; grid minor;

xlabel('Time'); ylabel('Servo');

legend('Input','Servo');

SECTION 3

my\_filename = 'Derivative5';

save( strcat( my\_filename , '.mat' ) ,'results');

SECTION 4

input = pos\_input;

output = pos\_servo;

dtmean = mean(dt);

data = iddata(output,input,dtmean);

g = tfest(data,2,0);

# 9 References

1. Escalona, L. “Electrical Circuits and Components.” ME 330 – Mechatronics. San Diego State University. Lecture
2. Escalona, L. “Analysis of Electrical Networks.” ME 330 – Mechatronics. San Diego State University. Lecture
3. Escalona, L. “Sensors and Actuators Used in Mechatronics Systems.” ME 330 – Mechatronics. San Diego State University. Lecture
4. Escalona, L. “Semiconductor Devices: The Field Effect Transmitter.” ME 330 – Mechatronics. San Diego State University. Lecture
5. Escalona, L. “Operational Amplifier Configurations.” ME 330 – Mechatronics. San Diego State University. Lecture
6. Escalona, L. “Electric Motors.” ME 330 – Mechatronics. San Diego State University. Lecture
7. Escalona, L. “Lab 5 Closed-Loop Servo Controller.” ME 330 – Mechatronics. San Diego State University. PDF